(19)日本国特許庁（ＪＰ）　(12)特　許　公　報（Ｂ２）　(11)特許番号

第2561801号

(45)発行日　平成８年(1996)12月11日　(24)登録日　平成８年(1996)９月19日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　9/46	３４０		Ｇ０６Ｆ　9/46	３４０Ｂ	

請求項の数16(全 9 頁)

(21)出願番号　特願平6－11510

(22)出願日　平成６年(1994)２月３日

(65)公開番号　特開平6－250853

(43)公開日　平成６年(1994)９月９日

(31)優先権主張番号　０２１９１８

(32)優先日　1993年２月24日

(33)優先権主張国　米国（ＵＳ）

(73)特許権者　390009531
インターナショナル・ビジネス・マシーンズ・コーポレイション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国10504、ニューヨーク州　アーモンク　（番地なし）

(72)発明者　ブレーク・ゴードン・フィッチ
アメリカ合衆国10805　ニューヨーク州、ニュー・ロッシェル、アイカード・レーン　１

(74)代理人　弁理士　合田　潔　（外２名）

審査官　吉岡　浩

最終頁に続く

(54)【発明の名称】　プロセス・スケジューリングの管理方法およびシステム

(57)【特許請求の範囲】

【請求項１】データ処理環境内の複数の制御コンテキスト間の処理ノードにおけるプロセス・スケジューリングを管理する方法において、

（ａ）前記複数の制御コンテキストの中の１つである所定の制御コンテキスト内でデータを実行するステップと、

（ｂ）スケジューリング・イベントの処理に応答して、前記ステップ（ａ）のデータ実行をもたらし、前記所定の制御コンテキスト内からプロセス・スケジューリングを評価するステップと、

（ｃ）前記複数の制御コンテキストの中の１つであり、前記プロセス・スケジューリング評価ステップ（ｂ）に準じた優先順位を有する再開制御コンテキスト内でデータの実行を再開するステップと、を含む、プロセス・スケジューリング管理方法。

【請求項２】前記プロセス・スケジューリング管理方法は、ノンプリエンプティブなデータ処理環境内で使用され、前記再開ステップ（ｃ）は前記所定の制御コンテキスト内でデータの実行を再開することを含み、この場合、前記所定の制御コンテキストが前記再開制御コンテキストとなる、請求項１記載の方法

【請求項３】前記データ処理環境は、多重処理ノードを有する並列データ処理環境から成り、該並列データ処理環境にある多重処理ノードの各々でプロセス・スケジューリングを管理する前記方法を使用する、請求項１記載の方法

【請求項４】前記ステップ（ｂ）の評価は、前記所定の制御コンテキスト内から実行可能なスケジューリング・コードを前記データ処理環境から呼び出すことを含む、

請求項１記載の方法。

【請求項５】前記プロセス・スケジューリング管理方法は、前記評価ステップ（ｂ）に続いて前記所定の制御コンテキストから異なる制御コンテキストにコンテキストを切り換えるコンテキスト・スイッチ・ステップをさらに含み、該異なる制御コンテキストは、前記再開ステップ（ｃ）の前記再開制御コンテキストであり、該異なる制御コンテキストは前記複数の制御コンテキストの中の１つである、請求項１記載の方法。

【請求項６】前記コンテキスト・スイッチ・ステップは、

前記所定の制御コンテキストをディスケジューリングするステップと、

前記評価ステップ（ｂ）に準じた優先順位を有する前記異なる制御コンテキストを復元するステップと、を含む請求項５記載の方法。

【請求項７】前記プロセス・スケジューリング管理方法は、複数のスケジューリング・イベントの各々に対して前記ステップ（ａ）～（ｃ）を繰り返すステップをさらに含む、請求項１記載の方法。

【請求項８】データ処理環境内の複数の制御コンテキスト間でプロセス・スケジューリングを管理し、所定の制御コンテキストを処理ノードで実行し、該所定の制御コンテキストは前記複数の制御コンテキストの中の１つであるプロセス・スケジューリングの管理方法において、

（ａ）スケジューリング・イベントを処理するステップと、

（ｂ）前記複数の制御コンテキストの１つを含む再開制御コンテキストを決定するために、前記所定の制御コンテキスト内からプロセス・スケジューリングを評価するステップとを含み、

（ｃ）前記プロセス・スケジューリング評価ステップ（ｂ）に応答して前記再開制御コンテキスト内でデータの実行を再開するステップ、を含む、プロセス・スケジューリング管理方法。

【請求項９】前記プロセス・スケジューリング管理方法は、ノンプリエンプティブなデータ処理環境内で使用され、前記再開制御コンテキストは、前記再開ステップ（ｃ）が前記プロセス・スケジューリング評価ステップ（ｂ）に応答して、前記所定の制御コンテキスト内でデータの実行を再開することを含むように該所定の制御コンテキストを有する、請求項８記載の方法。

【請求項１０】前記データ処理環境は、多重処理ノードを有する並列データ処理環境を構成し、該並列データ処理環境にある処理ノードの各々で、プロセス・スケジューリングを管理する前記方法を使用する、請求項８記載の方法。

【請求項１１】前記プロセス・スケジューリング管理方法は、前記評価ステップ（ｂ）に続いて前記所定の制御コンテキストから前記再開制御コンテキストにコンテキストを切り換えるコンテキスト・スイッチ・ステップをさらに含み、該再開制御コンテキストは、前記所定の制御コンテキストとは異なる前記複数の制御コンテキストの中の１つであり、該再開制御コンテキストは前記評価ステップ（ｂ）によって定められ所定の制御コンテキストに勝る優先順位を有する、請求項８記載の方法。

【請求項１２】前記コンテキスト・スイッチ・ステップは、

前記所定の制御コンテキストをディスケジューリングするステップと、

前記評価ステップ（ｂ）に準じた優先順位を有する前記異なる制御コンテキストを復元するステップと、を含む請求項１１記載のプロセス・スケジューリング管理方法。

【請求項１３】複数の制御コンテキスト間で処理ノードにおけるプロセス・スケジューリングを管理し、該複数の制御コンテキストの中の１つを有する所定の制御コンテキストを実行する、プロセス・スケジューリング管理システムにおいて、

コンテキスト・スケジューリング・イベントを処理する手段と、

前記コンテキスト・スケジューリング・イベントの処理手段に応答して、再開制御コンテキストを識別するために前記所定の制御コンテキスト内から処理ノードにおけるプロセス・スケジューリングを評価する手段であって、該再開制御コンテキストは前記複数の制御コンテキストの中の１つであり、他のすべての前記複数の制御コンテキストに勝るスケジューリング優先順位を有する評価手段と、

再開制御コンテキストを識別する前記プロセス・スケジューリング評価手段に応答して、前記再開制御コンテキスト内の前記処理ノードでデータの実行を再開する手段と、を備える、プロセス・スケジューリング管理システム。

【請求項１４】前記処理ノードは、並列データ処理環境の複数の処理ノードの中の１つであり、前記プロセス・スケジューリング管理システムは、該並列データ処理環境の前記処理ノードの各々でプロセス・スケジューリングを管理する手段を備える、請求項１３記載のシステム。

【請求項１５】前記再開制御コンテキストは、前記所定の制御コンテキストとは異なる前記複数の制御コンテキストの中の１つであることを決定する前記評価手段の決定に続いて、該所定の制御コンテキストから該再開制御コンテキストに切り換えるコンテキスト・スイッチャを備える、請求項１３記載のシステム。

【請求項１６】前記コンテキスト・スイッチャは、

前記所定の制御コンテキストをディスケジューリングする手段と、

前記処理ノードで前記再開制御コンテキストを復元する手段と、を備える、請求項１３記載のシステム

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、一般的にデータ処理環境、より詳細にはコンテキスト・スイッチ（context switching：プロセッサの処理対象を１つのタスクから別のタスクに切り換えること）を使用して、アプリケーション・プログラムの多重スレッド処理を容易にするようなデータ処理環境のプロセス・スケジューリング管理方法およびシステムに関する。

【０００２】

【従来の技術】標準のプロセッサ・アレイを並列に用いて実行するように記述されたプログラムが増加している。アプリケーション・プログラマが、これらのプロセッサを利用することができ、且つ各処理ユニットの効率的な共用を調整することができるようにソフトウェア・スーパバイザ・システムを記述することによって、全体的なプログラム処理の目標を達成することができる。このようなソフトウェア・スーパバイザは、従来技術では並列「マイクロカーネル（microkernel）」と呼ばれている。総合的には、これらのマイクロカーネルは、任意の単一カーネルまたはオペレーティング・システムによって生成されたアプリケーション・プログラム環境と同様の環境を生成する。

【０００３】並列マイクロカーネルは、周辺マシン環境の資源を管理するという点で、単一カーネルと類似しているが、両者の間には著しい相違点が存在する。具体的にいうと、単一カーネルは、競合するタスク・コンテキスト間で、マシン環境を時間的に公平に分割しようとする。一方、並列マイクロカーネルは、一般的に、単一の最終目的のため、すなわち並列アプリケーション・プログラムの個々のカーネル部を効率的に完了するために、各マシン環境の能力を効果的に引き出すように記述されている。並列マイクロカーネル内で、構成タスクは実際には決して競合しない。なぜならば、それらはこの単一の最終目的に向かって１つのアプリケーション・プログラム内で動作しているからである。

【０００４】制御コンテキストのスケジューリングおよびディスケジューリング（descheduling）は、すべてのカーネルの義務の中心である。制御コンテキストは、コンテキストの特性およびカーネルの命名規則により、「スレッド」、「プロセス」、または「タスク」といった異なった名称が与えられる。コンテキストは、生成されるまで、ブロックされて待機中になるまで、または割り込まれるまで、マシン環境の制御下におかれる。スケジューリング・アルゴリズムが、コンテキスト実行の優先順位をこれらの時に評価するのでこれらの事象は、ディスケジューリング・イベントと呼ばれる。ノンプリエンプティブ（nonpreemptive：すなわち処理が開始されたジョブが完了まで中断することなく処理されること）・スケジューリングにおいて、ユーザ・コンテキストは、割り込みの結果として決してスイッチされることはない。結局、すべてのプロセッサの割り込みは、割り込みの時にコンテキスト処理に戻る。

【０００５】ノンプリエンプティブ並列マイクロカーネルにおいて、アプリケーション・プログラマはコンテキスト・スケジューリングを制御する。これは、単一スレッド・モデルでプログラミングすることによって直接的に、または多重コンテキスト間で生成することによって間接的に行われる。前者の場合、マシン環境当たり単一の応用コンテキストのみが存在し、したがってプログラマの見地から、走行可能時にいつもスケジュールがされているはずである。後者の場合、プログラマは特別にブロッキング・カーネル呼び出しを生成または作成し、参加するユーザ・コンテキストのスケジューリングを制御する。いずれの場合も、マイクロカーネル・サービスに対する呼び出しが、走行しているコンテキストをブロックし、スケジューリング・アルゴリズムの評価をもたらす。いずれにせよ、コンテキスト・スイッチはしばしば冗長である。これは、マイクロカーネルが、スケジューリング・アルゴリズムの評価の後に、優先コンテキストが、まさに生成したコンテキストであるかブロックしたコンテキストであるかをしばしば決定するからである。

【０００６】

【発明が解決しようとする課題】しかしながら、ここに問題が存在する。標準的なカーネル・コンテキストのスケジューリングにおいて、完全なコンテキスト・スイッチは、スケジューリング・アルゴリズムを再評価する機会があればいつでも実行される。優先権が与えられた次のコンテキストが、生成しまたはブロックしたコンテキスト以外のコンテキストであるという可能性が高いことを想定する。並列マイクロカーネル環境において、これは厳しい想定である。実際に、あるコンテキストが、自身が継続的に再スケジューリングされて実行されることを認識すると、冗長なコンテキスト・スイッチが生じる可能性が高い。したがって、スケジューリング・アルゴリズムの評価によってコンテキスト・スイッチが冗長性であることがわかると、データ処理技術において、コンテキスト・スイッチの荷重なオーバヘッドに影響されないスケジューリング方法が必要とされることとなる。

【０００７】

【課題を解決するための手段】簡単に要約すれば、本発明の一側面として、データ処理環境の処理ノードで少なくとも２つの制御コンテキスト間のプロセス・スケジューリングを管理する方法を有する。この方法は、少なくとも２つの制御コンテキストの中の１つである所定の制御コンテキスト内でデータを実行するステップと、スケジューリング・イベントの処理に応答してデータの実行をもたらし、所定の制御コンテキスト内からプロセス・スケジューリングを評価するステップと、少なくとも２つの制御コンテキストの中の１つである再開制御コンテ

キスト内でデータの実行を再開するステップとを含む。拡張された実施例において、データ処理システムはノンプリエンプティブ（nonpreemptive）な並列データ処理システムを形成する。

【０００８】他の側面として、処理ノードにおける複数の制御コンテキスト間でプロセス・スケジューリングを管理するシステムを提供する。所定の制御コンテキストが、当該複数の制御コンテキストの中の１つであり処理ノードで実行されるものと想定される。この管理システムは、コンテキスト・スケジューリング・イベントを識別する処理手段を含む。コンテキスト・スケジューリング・イベントが識別されると、コンテキスト・スケジューラは、この所定の制御コンテキスト内から評価され、当該処理ノードに対して再開制御コンテキストを識別する。この再開制御コンテキストは、複数の制御コンテキストの中の１つであり、複数の走行可能な制御コンテキストのいずれにも勝るスケジューリングの優先順位を有する。実行再開手段は、スケジュール評価手段によって識別された再開制御コンテキスト内の処理ノードでデータの実行を再開するのにも提供される。

【０００９】要約すれば、データ処理環境内の多重制御コンテキスト間でプロセス・スケジューリングを管理する新規な技術が提供される。この技術は、プロセス・スケジューリング機能を個別の部分に分割することを含む。第１の機能は、処理制御コンテキスト内からの評価機能であり、コンテキスト・スイッチが保証されるか否かを決定するために使用される。第２に、コンテキスト・スイッチが保証されれば、コンテキスト・スイッチャが呼び出され、ディスケジューリング制御コンテキストの状態を保存し、優先制御コンテキストの状態を復元する。この第２の機能は、ディスケジューリング・コンテキスト内で、または特権コンテキストとして実行される。この管理技術は、単一のコンピュータ・ノードがノンプリエンプティブ処理スケジューリングを提供する単一プログラムによって使用される状況で最も有用である。スケジューリング機構は、冗長なコンテキスト・スイッチが呼び出されるどのような状況でも有用であり、これによって性能オーバヘッドをもたらす。この概念は、プリエンプティブ処理スケジューリングにも容易に拡張できる。

【００１０】

【実施例】図１は、並列データ処理環境１０の簡略化した実施例であり、処理ノード１２のアレイを有している。各処理ノード１２は、ハードウェア、ソフトウェアとハードウェアの組み合わせ、または主にソフトウェア（例えば仮想処理ノード）で実現される　総合すれば、処理ノード１２は、どのデータ処理機能が実行されるマシン環境を提供する。ある標準的な実施例において、各処理ノード１２は、ＣＰＵ、通信インターフェース、メモリおよびクロック機構を有する

【００１１】処理ノードのアレイ内において、通常、ノードは高い優先順位のメッセージと低い優先順位のメッセージによって通信を行う。一例として、図２に示すように、４つの処理ノード１２（または処理スタック）は、通信インターフェース１６に接続され、各々のノード間で双方向にデータを転送を行う。本実施例において、各処理ノード１２は、ソフトウェア構成１８およびハードウェア構成１７を有し、ともに処理スタックを形成する。ソフトウェア構成１８は、アプリケーション・プログラム２２および制御システムまたはカーネル２４を有する。各制御システムまたはカーネル２４は、大規模なシステムすなわち並列データ処理環境の一部分であることを本質的に理解している。カーネル２４は、処理ノード１２間で情報の流れを制御する。

【００１２】図３に示すように、並列データ処理環境にあるすべての処理ノード１２は、互いに関連した単一のアドレス空間「ＡＤＤＲ．０」、「ＡＤＤＲ．１」、「ＡＤＤＲ．２」、「ＡＤＤＲ．３」を有する。すなわち、完全なアドレス空間が多重スレッド化されている。各々のカーネル／アプリケーション・インターフェース１９は、他のすべての処理ノード１２で得られるグローバル関数と同じグローバル関数を、その処理ノードで利用できるようにする。所定のアドレス空間内でデータを実行することによって、そのアドレス空間に影響を及ぼすことができ、または通信インターフェース１６（図２）を通してそのデータでアドレスされたオブジェクトのアドレス空間に同等の効果を生成することができる。以下に示すコンテキスト・スケジューリングの方法／システムは、並列データ処理環境の単一アドレス空間内で処理性能を高めることを追及している（本発明は、アプリケーション・プログラムの多重スレッドを処理するために割り当てられた単一プロセッサの性能を向上させるようにも拡張できる）。

【００１３】オペレーティング・システムは、異なる制御コンテキスト（すなわち異なるプロセスおよび／またはスレッド）間で、あるオペレーティング・システムによって管理されるマシン環境を時間的に分割する。スケジューリング・イベントに到達すると、従来の方法では以下の３つのステップを順番に実行していた。すなわち、（１）現コンテキストのパラメータを保存することによって現在走行しているコンテキストをディスケジューリング（descheduling）する、（２）オペレーティング・システム内のスケジューリング・コードを用いて、次に走行するコンテキスト（オペレーティング・システムのコンテキストのロードが通常必要とされる）を決定する、（３）新しいコンテキストを（処理ノードで新しいコンテキストのパラメータをロードすることによって）スケジューリングすることの３つである

【００１４】この方法とは対象的に、本発明によれば、スケジューリング・コードはまず現コンテキスト内から

評価され、コンテキスト・スイッチが発生するような決定が（スケジューリング・コードを走行することによって）なされるときのみ、この現コンテキストがディスケジューリングされる　本質的に本発明は、スケジューリング機能を、現制御コンテキスト内から再び行われる評価段、および条件付きのコンテキスト・スイッチング段に分割することである。本発明の一実施例において、スケジューリング・コードは、どのような制御コンテキストによっても読み出し可能なサブルーチン・ライブラリのコード（すなわち、どのコンテキストが優先権を有するかを決定するために走行させなければならないプログラム・ルーチン）を提供することによって、現在走行中のコンテキスト内から実行可能であり、どのような再スケジューリング地点においても、たとえばサブルーチン・コールによってまたは割り込みによって、現在走行しているコンテキスト内から実行可能である（少数のパラメータはサブルーチン・コールまたは割込み時に必ず保存される一方で、大多数のパラメータはプロセスまたはスレッド・コンテキストがスイッチされるときはいつでも保存されなければならない）。

【００１５】コンテキスト・スイッチは、従来の「コンテキスト・スイッチャ」機能によって行われる。この機能には、通常、ディスケジューリングされたコンテキストの全状態を保存し、カーネルのスケジューリング構造を更新し、その後、到来する制御コンテキストの全状態を回復することが含まれる。コンテキスト・スイッチャは、たとえば実行するディスケジューリング・コンテキストを用いることにより、自身の制御コンテキストをもたなくてもよいし、またはスーパーバイザ・コールによって入力された特権コンテキストから実行されるようにしてもよい。

【００１６】図４および図５に、本発明によるコンテキスト・スケジューリングを管理する方法／システムの一実施例を示し、ノンプリエンプティブ・スケジューリング環境と関連して以下に示す。しかしながら本発明は、スケジューリング・コードで用いられたデータの構造が、スケジューリングされているコンテキスト内からアクセス（すなわち読み出し）される限り、プリエンプティブ・スケジューリングへ拡張するよう容易に一般化される。

【００１７】処理ノードにおいて、多数の制御コンテキストの中の１つを実行することによって処理を開始する（ステップ４０「所与のコンテキスト内で実行」）。スケジューリング・イベントを処理し（ステップ４２「スケジューリング・イベントを処理する」）、その後、実行コンテキストを随時（voluntarily）生成するか否かを初期問合せする（ステップ４４「実行コンテキストを随時生成するか？」）　Ｎｏならば、その後、走行中の制御コンテキストをブロックして待機させ、結合部「Ａ」４６を経て処理を図５の制御シーケンスに渡す

【００１８】引き続き図４において、実行制御コンテキストを随時生成する場合、次に、関連する走行準備（ready to run）待ち行列が空か否かを問合わせる（ステップ４８「走行準備待ち行列が空であるか？」）。実行制御コンテキストを随時生成しているときはいつでも、その優先順位が他の走行準備待ち行列よりも高いと確認されればすぐに実際に実行が継続される」したがって、走行準備待ち行列が空であれば、結合部５６から命令５８へ処理を受け渡し（ステップ５８「生成コンテキストに戻る」）、それによって、生成中のコンテキストに即時戻りを作成する。

【００１９】走行準備待ち行列が１つ以上の制御コンテキストを含んでいると、その後、スケジューリング・アルゴリズムを適用し、関連する走行準備待ち行列の先頭にある制御コンテキストとその生成中の生成制御コンテキストとの間で優先順位を決定する（ステップ５０「スケジューリング・アルゴリズムを使用」）。再び、たとえばサブルーチン・コールによって生成制御コンテキスト内からスケジューリング・アルゴリズムが適用される。生成制御コンテキストが依然として優先権を有するならば（ステップ５２「生成されたコンテキストが優先権を有するか？」）、結合部５６を経て命令５８で生成制御コンテキストに戻る（ステップ５８「生成コンテキストに戻る」）。これとは反対に、新しい制御コンテキストが優先権を有すれば、コンテキスト・スイッチャを呼び出し、この優先権を有する制御コンテキストに処理ノードを移動する、すなわちコンテキストが切り換えられる（ステップ５４「コンテキスト・スイッチャを呼び出す」）。

【００２０】図５を参照して、実行制御コンテキストがブロックされてあるイベントを待機中であれば、関連する走行準備待ち行列が空か否かを問合せる（ステップ６０「走行準備待ち行列は空か？」）。走行準備待ち行列が空でなければ、コンテキスト・スイッチャを呼び出し現在走行準備されている代替コンテキストの実行が好ましい（ステップ６２「コンテキスト・スイッチャを呼び出す」）。しかしながら、関連する走行準備待ち行列が空であれば、処理ノードは待機し（ステップ６６「スケジューリング・アルゴリズムを待機」）、その後、関連する走行準備待ち行列が空であるか否かを再び問合せる（ステップ６８「走行準備待ち行列は空か？」）。走行準備待ち行列が空であれば、プロセッサは結合部６４を経て待機状態に戻り、再び所定期間だけ待つ（ステップ６６「スケジューリング・アルゴリズムを待機」）。そのうちに、制御コンテキストが、走行準備状態になると、走行準備待ち行列の先頭に配置される　これが生じると、走行準備待ち行列の制御コンテキストは、ブロックされた制御コンテキストと同し制御コンテキストであることになる（ステップ７０「優先権を有するコンテキストがブロックされたコンテキストと同じか？」）　も

11

し同じであれば、その制御コンテキストに対して即時戻りを作成する（ステップ７４「最後にブロックされたコンテキストに戻る」）。さもなければ、コンテキスト・スイッチャを呼び出し、走行の準備された制御コンテキストに処理ノードを移動する（ステップ７２「コンテキスト・スイッチャを呼び出す」）。

【００２１】再び要約すれば、データ処理環境内の多重制御コンテキスト間でプロセス・スケジューリングを管理する新規な技術を提供する。この技術は、プロセス・スケジューリング機能を個別の部分に分割することを含む。第１の機能は、処理制御コンテキスト内からの評価機能であり、コンテキスト・スイッチが保証されるかどうかを決定するために使用される。第２の機能は、コンテキスト・スイッチが保証されれば、コンテキスト・スイッチャが呼び出され、ディスケジューリング制御コンテキストの状態を保存し、優先制御コンテキストの状態を復元する。この第２の機能は、ディスケジューリング制御コンテキスト内で、または特権コンテキストとして実行することができる。この管理技術は、単一のコンピュータ・ノードがノンプリエンプティブ処理スケジューリングを提供する単一プログラムによって使用される状況で最も有用である。このスケジューリング機構は、冗長なコンテキスト・スイッチが呼び出されるどのような状況でも有用であり、これによって性能オーバヘッドがもたらされる。この概念は、プリエンプティブ処理スケジューリングにも容易に拡張できる。

【００２２】

【発明の効果】以上説明したように、本発明によれば、コンテキスト・スイッチングを行う際に、処理制御コンテキスト内からこれが評価されるので、コンテキスト・スイッチの荷重なオーバヘッドに影響を受けないという効果が得られる。

【００２３】多重制御コンテキストの中の１つである所定の制御コンテキストは、データ処理環境１０内の処理

12

ノード１２で実行される。この方法は、スケジューリング・イベントを処理することを含んでおり、実行制御コンテキスト内から処理スケジューリングを評価して多重制御コンテキストの中の１つである再開制御コンテキストを決定し、かつ所定の制御コンテキストで起こっているプロセス・スケジューリングに応答して再開制御コンテキストでデータの実行を再開する。コンテキスト・スイッチは、再開制御コンテキストが所定の制御コンテキスト以外のコンテキストであるのを処理スケジューリングが決定するときのみ用いられる。この技術はノンプリエンプティブな並列データ処理環境で特に有利である。

【図面の簡単な説明】

【図１】本発明の並列データ処理環境の一実施例を示す図である。

【図２】本発明の並列データ処理環境において、通信インターフェースを通して相互接続された幾つかの処理ノードを示す図である。

【図３】図２の並列処理ノードのアドレス空間割り当ての一実施例を示す図である。

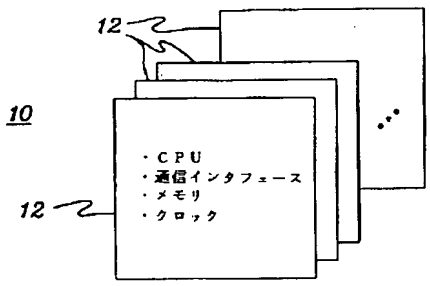
【図４】本発明に準じたコンテキスト・スケジューリングを管理する方法の一実施例を示すフロー図である。

【図５】本発明に準じたコンテキスト・スケジューリングを管理する方法の一実施例を示すフロー図である。

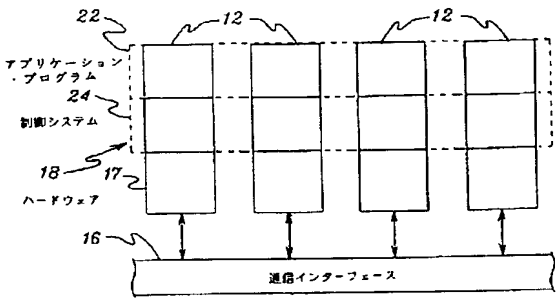
【符号の説明】

１０　データ処理環境
１２　処理ノード
１６　通信インターフェース
１７　ハードウェア成分
１８　ソフトウェア成分
１９　カーネル／アプリケーション・プログラム・インターフェース
２２　アプリケーション・プログラム
２４　制御システムまたはカーネル

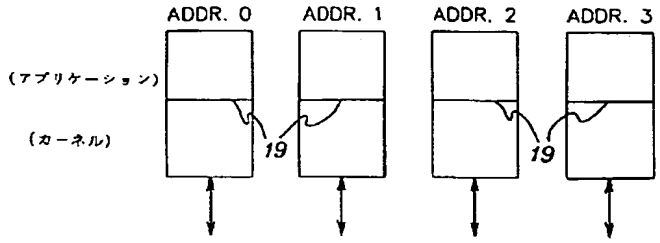
【図１】



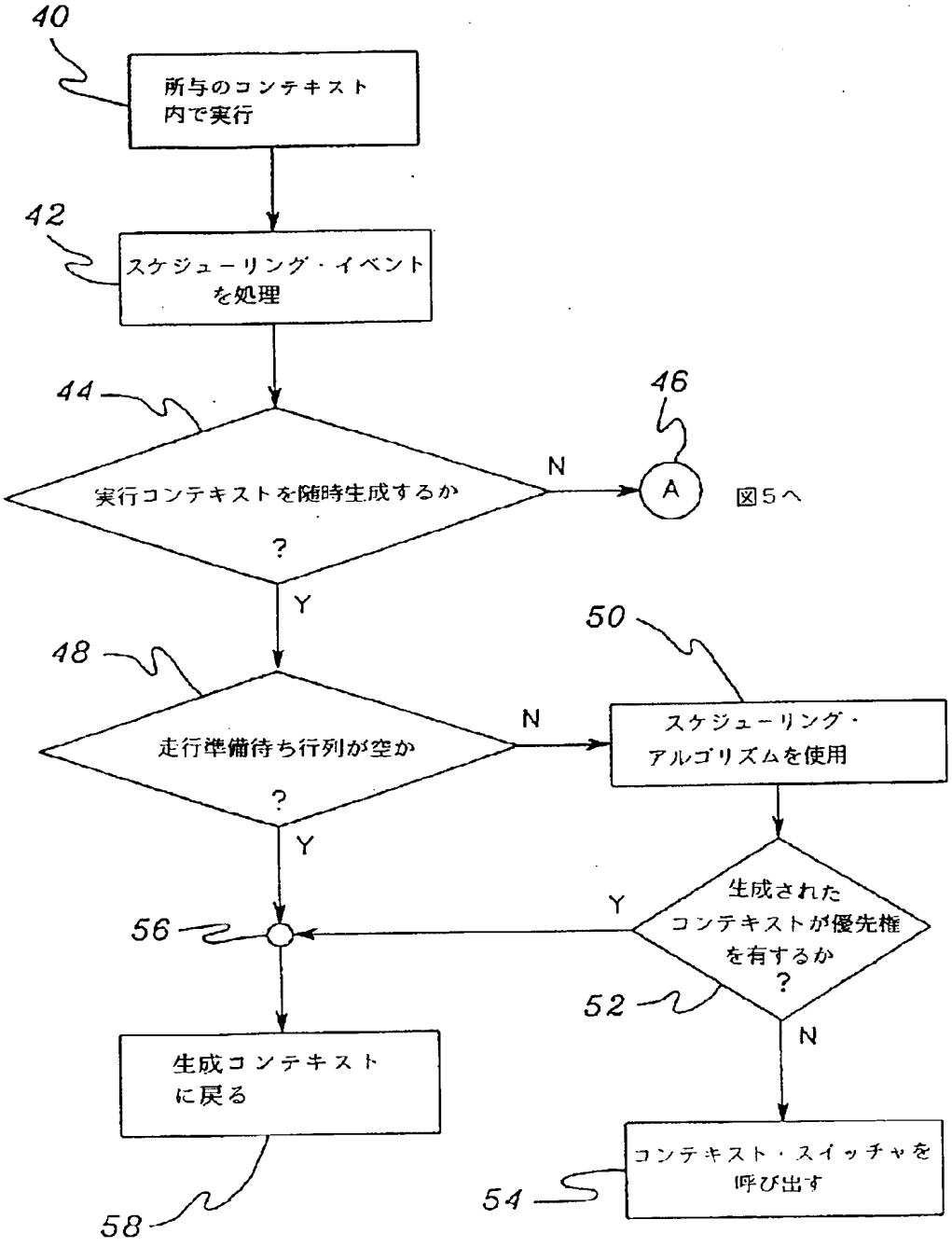
【図２】



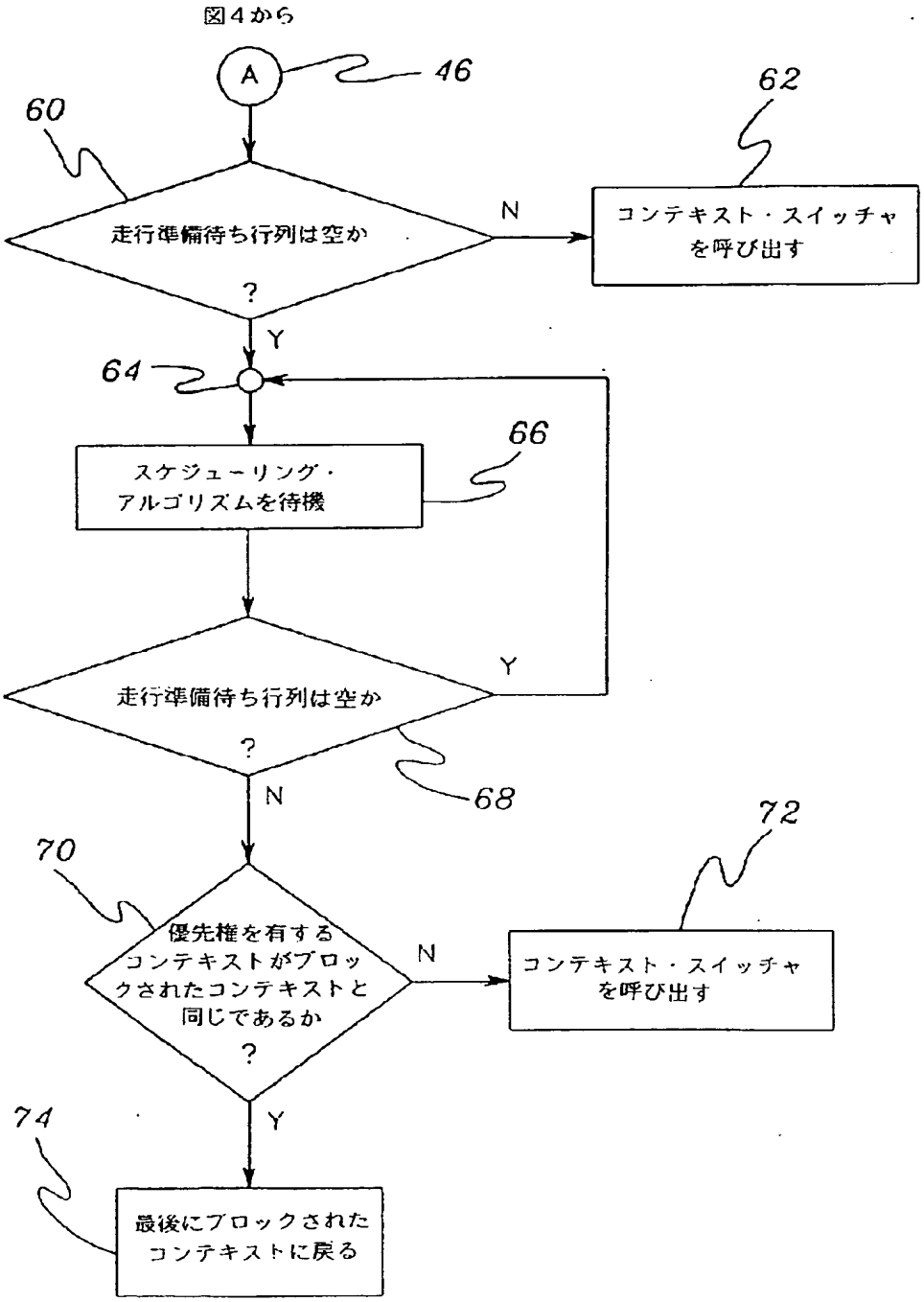
【図３】



【図4】


40
所与のコンテキスト内で実行
42
スケジューリング・イベントを処理
44
実行コンテキストを随時生成するか？
N
46
A
図5へ
Y
48
走行準備待ち行列が空か？
N
50
スケジューリング・アルゴリズムを使用
Y
生成されたコンテキストが優先権を有するか？
Y
56
52
N
生成コンテキストに戻る
コンテキスト・スイッチャを呼び出す
54
58

【図５】



フロントページの続き

(72) 発明者　マーク・エドウィン・ギアンハバ
アメリカ合衆国10533－1210　ニューヨーク州、アービントン、アパートメント・ジーティー－14、ノース・ブロードウェイ　140